On Unrelated Machine Scheduling with Precedence Constraints

Jeffrey Herrmann, Jean-Marie Proth, Nathalie Sauer

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

On the Unrelated Machine Scheduling with Precedence Constraints

Jeffrey Herrmann - Jean-Marie Proth
Nathalie Sauer

N° 2773
Janvier 1996

PROGRAMME 5

ON UNRELATED MACHINE SCHEDULING WITH PRECEDENCE CONSTRAINTS

Jeffrey HERRMANN*, Jean-Marie PROTH* and **, Nathalie SAUER**

ABSTRACT:
In this paper, we consider the problem of scheduling tasks on unrelated parallel machines. Precedence constraints exist between the tasks, but their number is limited compared with the number of tasks. We propose a number of heuristics in order to find near-optimal solutions to the problem. Empirical results show that the heuristics are able to find very good approximate solutions.
KEYWORDS: Parallel resources, Scheduling, Makespan.

ORDONNANCEMENT SUR MACHINES PARALLELES DE TYPES DIFFERENTS AVEC CONTRAINTES DE PRECEDENCE SUR LES TACHES

Jeffrey HERRMANN*, Jean-Marie PROTH* et **, Nathalie SAUER**

RESUME :
Dans ce papier, nous examinons le problème d'ordonnancement de tâches sur des machines parallèles différentes, sachant qu'un ordre partiel existe sur l'ensemble des tâches. Nous proposons plusieurs approches heuristiques. Les résultats numériques obtenus montrent que les heuristiques nous permettent d'atteindre d'excellentes solutions.
MOTS-CLEFS : Ressources parallèles, Ordonnancement, Makespan.

* Institute for Systems Research, University of Maryland, College Park, MD 20742, USA

** INRIA-Lorraine, Technopôle Metz 2000, 4 rue Marconi, F-57070 Metz, France

1. Introduction

The problem to be solved can be formulated as follows. A finite set of tasks needs to be performed by a set of workers, and certain tasks cannot begin until the others are completed. The time required to perform each task varies from worker to worker. We are interested in assigning the tasks to each worker and scheduling the tasks in such a way that the time needed to finish all of the work is minimized. In other words, we wish to minimize the maximal tasks completion time, also called the makespan.

Such a problem typically occurs in an office or project management environment, where the resources are people who have different skills: it is why the problem at hand is often called the Office Scheduling Problem (OSP for short). Due to the various skills available in an office the time necessary to complete a task can greatly vary from worker (resource) to worker. The tasks are the work that needs to be done during a specific period (a day for instance). Minimizing the makespan ensures that all of the tasks are done as soon as possible, and hopefully by the end of the assigned period. Furthermore, reducing the makespan leads to working periods which are not too different from worker to worker.

The precedence constraints reflect the fact that some of the tasks are related, although the number of such relationships may be quite small compared to the total number of tasks.

This problem is one of the unrelated machines scheduling. Even without precedence constraints, minimizing the makespan is a NP-complete problem. However, a number of researchers have proposed an analyzed heuristics for this problem.

Potts (1985) uses a linear programming relaxation of the assignment problem formulation. This problem can be solved to assign most of the tasks to machines. However, the solution has up to m-1 tasks divided between machines. Given the partial solution from the LP, the heuristic uses an exponential-time exhaustive search to determine the best assignment of these tasks. The worst-case relative performance of the heuristic is 2. If m = 2, an O(n) heuristic is presented; this heuristic has worst-case relative performance of 1.5.

Ibarra and Kim (1977) present a number of greedy heuristics similar to list scheduling. The worst-case relative performance of these heuristics is m, but the heuristics are polynomial. If m = 2, the worst-case performance is $(\sqrt{5}+1) / 2$.

Davis and Jaffe (1981) order on each machine all of the tasks by their efficiency, which is defined as the minimum processing time for the task divided by the processing time of this worker. Their heuristic schedules on the next available machine the most efficient task until the efficiency for all remaining tasks on the machine is too low. The effort of the heuristic is O(mn log n). The worst-case relative performance is $2.5\sqrt{m}$.

Lenstra, Shmoys and Tardos (1990) use a sequence of linear programs to find the smallest deadline for which the linear programming relaxation has a feasible schedule. In the second phase, a bipartite matching problem is solved to assign the unscheduled jobs to machines. The worst-case relative performance of this polynomial-time-heuristic is 2.

Hariri and Potts (1991) compare a number of heuristics for the problem. They consider two-phase heuristics which start with an LP relaxation and then apply a matching or the earliest completion time heuristic (ECT) to assign the unscheduled jobs. They also consider the ECT procedure itself, proving that its worst-case relative performance is $1 + \log_2 n$. Finally, they consider reassignment and interchange heuristics to improve a schedule constructed by one of the above heuristics. They conclude that the use of improvement techniques with the ECT heuristic gives satisfactory solutions, though slightly better results can be achieved by using the two-phase heuristics with the improvement techniques.

Van de Velde (1993) uses a surrogate relaxation problem to derive a lower-bound. The search for the best lower bound is an ascent direction algorithm. An approximation algorithm based on the dual problem is presented and compared to solutions found with a branch-and-bound and other heuristics. A branch-and-bound over the job assignment is able to solve some quite large problems using less than 100 000 nodes.

In conclusion, it appears that the ECT heuristic with improvements provides simple but good approximation for the problem. If better solutions are required, the duality-based algorithm would be preferred, since it does not required an enumeration or a sequence of linear programs.

The research on problems with precedence constraints has been restricted largely to identical parallel machines. Ullman (1975) proves that even if all the jobs have length one, the problem is NP-complete, although the problem can be solved in polynomial time if the precedence constraints form a tree (Hu,

1961). Graham (1966) considers the worst-case performance of list scheduling in the presence of precedence constraints, which is 2 - 1/m. Du, Leung and Young (1989) show that the two-machine problem with arbitrary processing times and tree precedence constraints is strongly NP-complete. Cheng and Sin (1990) review a number of results on the worst-case performance of list scheduling and highest-level-first approximation algorithms for minimizing makespan on identical parallel machines. Hoitmont et al. (1990) present a Lagrangian relaxation technique for the problem of minimizing total weighted quadratic tardiness on identical machines with precedence constraints. Outside of these approaches, however, few approximation algorithms have been proposed, and none consider non-identical machines.

In this paper, we address the lack of heuristics for parallel resources scheduling with precedence constraints, the resources being different.

The problem is presented in section 2. In section 3, we propose a straightforward heuristic which assigns tasks to resources and schedules the tasks simultaneously. Section 4 is devoted to twofold heuristics: the first step consists of assigning tasks to resources, and the second step schedules the tasks on the resources, taking into account the precedence constraints. Two approaches are proposed to perform the second step of the heuristic. In section 5, we propose several numerical examples and evaluate the performances of the algorithms proposed in this paper. Section 6 is the conclusion.

2. Problem formulation

The problem can be formulated as follows. There exists a set of m workers W_j, $j = 1,\ldots,m$, and a set of n tasks T_i, $i = 1,\ldots,n$. Task T_i requires time $p_{i,j}$ when performed by W_j. We denote by $O(T_i)$ the task, if any, which is the immediate successor to T_i. Similarly, we denote by $O^{-1}(T_i)$ the task, if any, which is the immediate predecessor to T_i. $O(T_i)$ cannot begin until T_i is completed. $O^2(T_i) = O(O(T_i))$, and $O^q(T_i)$ is similarly defined for $q > 2$. Each task must be performed by someone, and each worker performs at most one task at a time. A worker which begins a task should finish it, without a break. If C_i is the completion time of task T_i, the goal is to find a schedule such that $\max_{i\in\{1,\ldots,n\}} C_i$ is minimal.

3. Heuristic H1

This heuristic assigns the tasks to the resources and defines the schedule simultaneously. The basic principle of this heuristic is very simple: it consists of scheduling at each iteration the task which could lead to a late schedule of some tasks in the future.

Assuming that the predecessors of T_i (if any) have been scheduled with possibly other tasks, we denote by $\mu_{i,j}$ the smallest real such that:

(i) $$\mu_{i,j} \geq \begin{cases} 0 & \text{if } O^{-1}(T_i) = \varnothing \\ \mu_{i^*,j^*} + p_{i^*,j^*} & \text{where } T_{i^*} = O^{-1}(T_i) \text{ and } W_{j^*} \text{ is the worker to whom } T_{i^*} \text{ has been assigned} \end{cases}$$

(ii) $\left[\mu_{i,j}, \mu_{i,j} + p_{i,j}\right]$ is an idle period for W_j.

$\mu_{i,j}$ is the earliest time when W_j can start task T_i.

Note that $\mu_{i,j} + p_{i,j}$ may be smaller than the completion time of the tasks which have already been scheduled for W_j.

The following notations will also be used:

$T_{i(q)} = O^q(T_i)$ with $i(0) = i$

$W_{j(q)}$ is the worker to which $T_{i(q)}$ has been assigned

Let E_k be the set of unscheduled tasks at the beginning of the k-th iteration, and $F_k \subset E_k$ the set of tasks belonging to E_k and which are not successors of a task of E_k.

For each $T_i \in F_k$, we apply the algorithm presented hereafter.

1. <u>For j = 1...,m:</u>

   1.1. Assign task T_i to W_j and choose $\mu_{i,j}$ as the starting time of T_i. Note that $i(0) = i$ and $j(0) = j$.

   1.2. Set $\theta_{i,j} = p_{i,j} + \mu_{i,j}$.

   1.3. Let N be the number of successors of T_i.

1.3.1. If N = 0, go to 2.

1.3.2. If N > 0, for q = 1...,N:

1.3.2.1. Select j* such that $p_{i(q),j^*} + \mu_{i(q),j^*} = \underset{k\in\{1,\ldots,m\}}{Min}\left[p_{i(q),k} + \mu_{i(q),k}\right]$

1.3.2.2 Set $j(q) = j^*$.

1.3.3. Set $\theta_{i,j} = \mu_{i(N),j(N)} + p_{i(N),j(N)}$.

2. Compute j(i) such that:

$$\theta_{i,j(i)} = \underset{j\in\{1,\ldots,m\}}{Min} \theta_{i,j} \qquad (1)$$

We consider that the task which could lead to a late schedule of some tasks in the future is T_{i^*}, where i* is such that:

$$\theta_{i^*,j(i^*)} = \underset{i\in F_k}{Max} \theta_{i,j(i)} \qquad (2)$$

We assign T_{i^*} to $W_{j(i^*)}$ and fix its starting time at $\mu_{i^*,j(i^*)}$. We then start the next iteration. We stop when $F_k = \varnothing$.

Let us consider the following example which concerns two workers and seven tasks.

Example

The processing times are given in Table 1.

Table 1: Processing times

	T_1	T_2	T_3	T_4	T_5	T_6	T_7
W_1	3	4	8	2	5	9	3
W_2	9	5	2	6	10	4	8

Furthermore we have to consider the following partial order:

$O(T_1) = T_3$; $O(T_3) = T_7$; $O(T_2) = T_6$

Thus:

$F_1 = \{T_1, T_2, T_4, T_5\}$

We obtain, according to (1):

$\theta_{1,1} = 8$; $\theta_{1,2} = 14$; and thus $j_1 = 1$ and $\theta_{1,j1} = 8$

$\theta_{2,1} = 8$; $\theta_{2,2} = 9$; and thus $j_2 = 1$ and $\theta_{2,j2} = 8$

$\theta_{4,1} = 2$; $\theta_{4,2} = 6$; and thus $j_4 = 1$ and $\theta_{4,j4} = 2$

$\theta_{5,1} = 5$; $\theta_{5,2} = 10$; and thus $j_5 = 1$ and $\theta_{5,j5} = 5$

We thus assign T_1 to W_1 at the earliest. T_1 starts on W_1 at time 0 and is completed at time 3.

We then start the second iteration with $F_2 = \{T_2, T_3, T_4, T_5\}$.

$\theta_{2,1} = 11$; $\theta_{2,2} = 9$; and thus $j_2 = 2$ and $\theta_{2,j2} = 9$

$\theta_{3,1} = 14$; $\theta_{3,2} = 8$; and thus $j_3 = 2$ and $\theta_{3,j3} = 8$

$\theta_{4,1} = 5$; $\theta_{4,2} = 6$; and thus $j_4 = 1$ and $\theta_{4,j4} = 5$

$\theta_{5,1} = 8$; $\theta_{5,2} = 10$; and thus $j_5 = 1$ and $\theta_{5,j5} = 8$

We assign T_2 to W_2. T_2 will start at time 0 and will be completed at time 5.

The third iteration starts with $F_3 = \{T_3, T_4, T_5, T_6\}$.

$\theta_{3,1} = 14$; $\theta_{3,2} = 10$; and thus $j_3 = 2$ and $\theta_{3,j3} = 10$

$\theta_{4,1} = 5$; $\theta_{4,2} = 11$; and thus $j_4 = 1$ and $\theta_{4,j4} = 5$

$\theta_{5,1} = 8$; $\theta_{5,2} = 15$; and thus $j_5 = 1$ and $\theta_{5,j5} = 8$

$\theta_{6,1} = 14$; $\theta_{6,2} = 9$; and thus $j_6 = 2$ and $\theta_{6,j6} = 9$

We assign T_3 to W_2. T_3 will start at time 5 and will end at time 7.

The fourth iteration starts with $F_4 = \{T_4, T_5, T_6, T_7\}$.

$\theta_{4,1} = 5$; $\theta_{4,2} = 13$; and thus $j_4 = 1$ and $\theta_{4,j4} = 5$

$\theta_{5,1} = 8$; $\theta_{5,2} = 17$; and thus $j_5 = 1$ and $\theta_{5,j5} = 8$

$\theta_{6,1} = 14$; $\theta_{6,2} = 11$; and thus $j_6 = 2$ and $\theta_{6,j6} = 11$

$\theta_{7,1} = 10$; $\theta_{7,2} = 15$; and thus $j_7 = 1$ and $\theta_{7,j7} = 10$

We assign T_6 to W_2. T_6 starts at time 7 and ends at time 11.

The fifth iteration starts with $F_5 = \{T_4, T_5, T_7\}$.

$\theta_{4,1} = 15$; $\theta_{4,2} = 17$; and thus $j_4 = 1$ and $\theta_{4,j4} = 5$

$\theta_{5,1} = 8$; $\theta_{5,2} = 21$; and thus $j_5 = 1$ and $\theta_{5,j5} = 8$

$\theta_{7,1} = 10$; $\theta_{7,2} = 19$; and thus $j_7 = 1$ and $\theta_{7,j7} = 10$

We assign T_7 to W_1. T_7 starts at time 7 and ends at time 10.

The sixth iteration starts with $F_6 = \{T_4, T_5\}$.

$\theta_{4,1} = 5$; $\theta_{4,2} = 17$; and thus $j_4 = 1$ and $\theta_{4,j4} = 5$

$\theta_{5,1} = 15$; $\theta_{5,2} = 21$; and thus $j_5 = 1$ and $\theta_{5,j5} = 15$

We assign T_5 to W_1. T_5 starts at time 10 and ends at time 15.

The last iteration starts with $F_7 = \{T_4\}$.

$\theta_{4,1} = 5$; $\theta_{4,2} = 17$; and thus $j_4 = 1$ and $\theta_{4,j4} = 5$

We assign T_4 to W_1. T_4 starts at time 3 and ends at time 5.

The schedule is represented in Figure 1 and the makespan is 15.

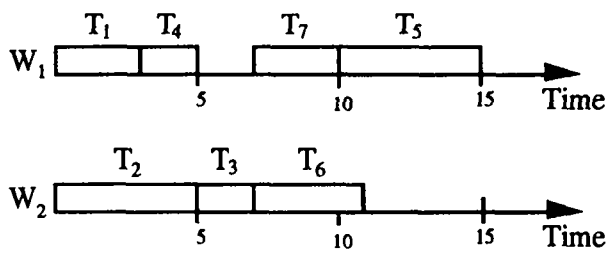

Fig. 1: Schedule when applying H1

4. Twofold heuristics

In this section, we present two twofold heuristics. In the first part of both heuristics, a branch-and-bound approach is used to assign tasks to workers. Two different approaches are used in the second part. The first one consists of applying H1 (see section 3) while keeping the assignment of tasks to workers as computed in the first part of the algorithm. The second approach consists of improving locally a feasible schedule by reducing the length of a critical path. Both algorithms take advantage of simulated annealing to improve the solution given as the result of the second part.

4.1. Part 1: Assignment of tasks to workers (Algorithm ASS)

At this level, we do not consider the precedence constraints. The objective is to minimize the makespan, i.e. to minimize $\max_{i\in\{1,\ldots,n\}} C_i$. To reach this goal, we utilize a branch-and-bound (B&B) approach. In the B&B tree, the descendants of a node represent the assignment of an unassigned task to the different workers. Thus, each node has m descendant nodes. The tasks are taken into account in the decreasing order of $\sum_{j=1}^{m} p_{i,j}$. More precisely, the tasks which are taken into account first are those having the greatest average processing time. Let us assume that the order obtained is $T_{i1}, \ldots, T_{in}$. The corresponding B&B tree is showed in Figure 2.

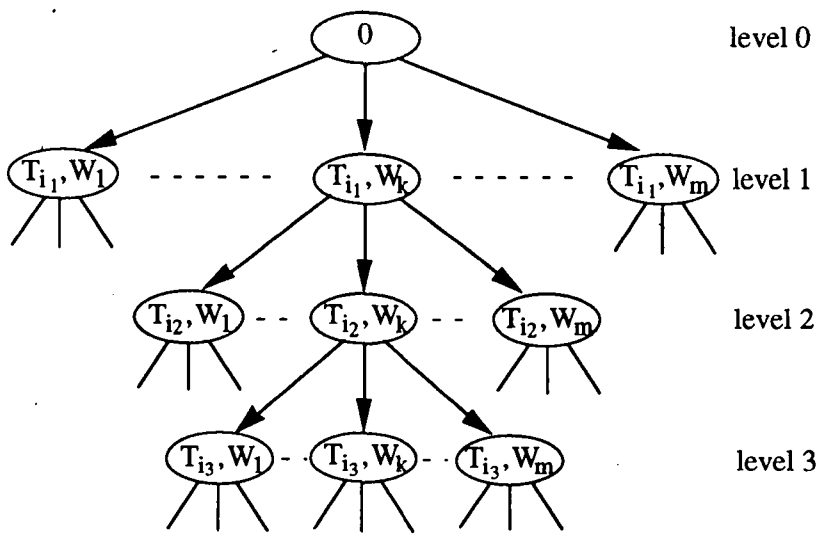

Fig. 2: The branch-and-bound tree

In this approach, we compute a lower bound of the objective function at each node. If the lower bound is greater than the current upper bound, then we stop generating descendants of this node, since we know that such descendants cannot lead to an optimal solution. Otherwise, we compute a new upper bound and continue the construction of the B&B tree.

Note that we refresh the upper bound at each node, if the node may lead to an optimal solution.

We will now explain how to compute the lower and upper bounds. Note that the closer the lower bound to the upper bound, the more efficient the B&B approach.

a. Computation of the lower bound

The lower bound is derived from the solution of the following linear programming (LP) problem.

Let E be the set of tasks which are not yet assigned to a worker when considering a node. We denote by μ_j the ending time of the last task assigned to W_j for $j = 1,\ldots,m$. Initially, $\mu_j = 0$. The task corresponding to the node under consideration is supposed to be assigned, and thus does not belong to E.

Let, for any $T_i \in E$:

$$x_{i,j} = \begin{cases} 1 & \text{if } T_i \text{ is assigned to } W_j \\ 0 & \text{otherwise} \end{cases}$$

In that case, the makespan, knowing the tasks which are already assigned, is given by:

$$\max_{j\in\{1,\ldots,m\}} \left(\mu_j + \sum_{i/T_i \in E} x_{i,j}\, p_{i,j} \right) \tag{3}$$

under the constraints:

$$\sum_{j=1}^{m} x_{i,j} = 1 \quad \text{for any i such that } T_i \in E \tag{4}$$

$$x_{i,j} \in \{0,1\} \text{ for any i such that } T_i \in E \text{ and for } j = 1,\ldots,m \tag{5}$$

The problem consists of minimizing (3), which represents the time when all the workers are completed their work. Constraints (4) are introduced to make sure that each task is assigned to one and only one worker.

This problem can be rewritten as:

$$\textbf{Min } z \tag{6}$$

s.t.

$$z \geq \mu_j + \sum_{i/T_i \in E} x_{i,j}\, p_{i,j} \quad \text{for } j = 1,\ldots,m \tag{7}$$

$$\sum_{j=1}^{m} x_{i,j} = 1 \quad \text{for any i such that } T_i \in E \tag{8}$$

$$x_{i,j} \in \{0,1\} \text{ for any i such that } T_i \in E \text{ and for } j = 1,\ldots,m \tag{9}$$

This is a mix LP problem.

Let us relax constraints (9) by replacing them by $x_{i,j} \geq 0$. Then the problem becomes a real LP problem, and its solution is a lower bound of the solutions corresponding to the node under consideration. It is the lowest bound we use at each node of the B&B tree.

b. Computation of the upper bound

Two upper bounds are computed, and we keep the smallest one.

b_1. Upper bound derived from the solution of an LP problem

We consider the solution of problem (6-9) where constraints (9) have been relaxed by replacing them by $x_{i,j} \geq 0$. Let $x^*_{i,j}$ be the solution of this real LP problem.

If, for each i such that $T_i \in E$:

(i) We compute $x^*_{i,j^*} = \underset{j\in\{1,\ldots,m\}}{\textbf{Max}}\ x^*_{i,j}$.

(ii) We set:

$$z_{i,j^*} = 1$$

$z_{i,j} = 0$ for $j \neq j^*$

then (see criterion (3)):

$$\underset{j\in\{1,\ldots,m\}}{\mathbf{Max}}\left(\mu_j + \sum_{i/T_i\in E} z_{i,j}\, p_{i,j}\right)$$

is an upper bound knowing the tasks already assigned at the node under consideration.

b_2. Upper bound obtained using a heuristic

This heuristic is heuristic H1, except that we do not consider the precedence constraints and we only take into account the tasks which are not yet been assigned, i.e. the set of tasks E. We denote by μ_j, $j = 1,\ldots, m$, the earliest time at which worker W_j completes the tasks which have been assigned to it previously (including the tasks corresponding to the node under consideration).

The algorithm applied to obtain the upper bound can be summarized as follows.

Algorithm UB

Let E_k be the set of tasks which have not been scheduled at the beginning of the k-th iteration. We set $E_1 = E$.

1. <u>For each i such that $T_i \in E_k$</u>

   1.1. For $j = 1$ to m, set $\theta_{i,j} = \mu_j + p_{i,j}$

   1.2. Compute j(i) such that:

   $$\theta_{i,j(i)} = \underset{j\in\{1,\ldots,m\}}{\mathbf{Min}}\ \theta_{i,j}$$

2. We consider task T_{i^*}, where i^* is such that:

$$\theta_{i^*,j(i^*)} = \underset{i/T_i\in E}{\mathbf{Max}}\ \theta_{i,j(i)}$$

We assign T_{i^*} to $W_{j(i^*)}$ and fix its starting time at $\mu_{i^*,j(i^*)}$. We then start the next iteration. We stop when $E_k = \emptyset$.

The upper bound corresponding to the node under consideration is:

$$\theta^* = \max_{i\in\{1,\ldots,n\}} \left\{\mu_{i,j(i)} + p_{i,j(i)}\right\}$$

The new upper bound will be the minimum value among the previous upper bound (if any, that is if the node under consideration is not the root of the B&B tree), and the two upper bounds computed in subsections b_1 and b_2. Note that the upper bound presented in b_1 takes advantage of the computation of the lower bound, and that algorithm UB, which computes the second upper bound, is very fast.

4.2. Part 2: Scheduling the tasks on the resources

As we mentioned previously, two approaches are used to schedule the tasks on the resources. The first one, which is referred to as H2, consists of applying algorithm H1, modified in order to keep the assignment of tasks to workers as computed in part 1 of the algorithm. The second one is referred to as CP. It consists of improving step by step an initial solution which can be generated at random.

4.2.1. Approach H2

This approach consists of scheduling the tasks to be performed by the workers by giving the priority to the tasks which could lead to a late schedule of some tasks in the future. We know the assignment of the tasks to the workers, and we denote by $W_{j(i)}$ the worker to which task T_i is assigned. The other notations are those used previously.

1. Set $E = \{T_1,\ldots,T_n\}$.
2. Set $F = \{T_i / T_i \in E \text{ and } \exists\ T_k \in E \text{ s.t. } T_i = O(T_k)\}$.
3. For all i such that $T_i \in F$:
   - 3.1. Let $\mu_{i,j(i)}$ be the earliest time when $W_{j(i)}$ can start performing T_i (see definition in section 3).
   - 3.2. Compute $\theta_i = \mu_{i,j(i)} + p_{i,j(i)}$.

3.3. Let N be the number of successors of T_i. $\theta_i = \mu_{i(N),j(i(N))}$ is computed using $\mu_{i(N-1),j(i(N-1))}$ which, in turn, is computed using $\mu_{i(N-2),j(i(N-2))}$, and so on until $\mu_{i(N),j(i(N))}$ which is derived from the state of the system, assuming that T_i is scheduled.

4. End of loop i.

5. Compute i* such that:

$$\theta_{i*} = \underset{i/T_i \in F}{\mathbf{Max}}\, \theta_i$$

6. Schedule T_{i*} on $W_{j(i*)}$ and fix its starting time at $\mu_{i*,j(i*)}$.

7. Set $E = E \setminus \{T_{i*}\}$.

8. If $E \neq \emptyset$, go to 2, else stop.

4.2.2. Approach CP

This approach is more complex than the previous one. In this section, we use the notations introduced previously.

We also denote by S_i the starting time of T_i and by C_i the completion time of T_i.

a. The critical path

The set $\{T_1,...T_n\}$ of tasks being scheduled, we call critical path for this schedule a sequence:

$$U = < T_{i1}, \ldots, T_{is} >$$

of tasks ($s \leq n$) such that:

1. $S_{i1} = 0$

2. $C_{ij} = S_{ij+1}$ for $j = 1,...,s-1$

3. $C_{i_s} = \underset{i\in\{1...,n\}}{\mathbf{Max}}\, C_i$ (makespan)

C_{is} is the makespan, also called the length of the critical path. It is clear that a necessary (but not sufficient!) condition to reduce the makespan is to reduce the length of the critical path. This leads to the concept of **degree of freedom** introduced hereafter.

b. Degree of freedom

We call degree of freedom of task T_i, and we denote it by $D(T_i)$, the time T_i which can be delayed without increasing the makespan. Formally:

$$D(T_i) = \mathit{Min}\left(C_{i_s} - C_i, D(T_{i(1)}) + S_{i(1)} - C_i, D(T_{i+}) + S_{i+} - C_i\right) \quad (10)$$

where:

$S_{i(1)}$ is the starting time of $O(T_i)$, if its exists,

T_{i+} is the task, if it exists, which is performed just after T_i by the same worker $W_{j(i)}$,

S_{i+} is the starting time of T_{i+},

$D(T_{i(1)}) + S_{i(1)} - C_i$ vanishes from (10) if $(T_{i(1)})$ does not exist.

$D(T_{i+}) + S_{i+} - C_i$ vanishes from (10) if T_{i+} does not exist, that is if T_i is the last task performed by $W_{j(i)}$.

The degrees of freedom are computed recursively, from the last tasks performed by the same worker to the first ones and, inside this order, from the last tasks to the first ones in the same sequence. They are used to select the tasks which should be permuted to reduce the makespan.

c. Specification of the tasks to be permuted

Two tasks T_i and T_j can be permuted without increasing the makespan if:

(i) T_i belongs to the critical path,

(ii) T_j is performed by the same worker $W_{j(i)}$ as T_i, and immediately before T_i,

(iii) it is possible to permute T_j and T_i without violating the precedence constraints,

(iv) $$C_j^1 \leq \mathit{Min}\left(D(T_{j(1)}) + S_{j(1)}, D(T_{i+}) + S_{i+}\right) \quad (11)$$

where:

C_j^1 is the completion time of T_j after permuting T_i and T_j and starting them at the earliest;

T_{i+} is the task which follows immediately T_i in the schedule of the tasks assigned to $W_{j(i)}$ before permuting T_i and T_j;

S_{i+} is its starting time.

If $O(T_j) = \varnothing$, then $D(T_{j(1)}) + S_{j(1)}$ vanishes in the second member of (11). If T_{i+} does not exist, then $D(T_{i+}) + S_{i+}$ vanishes in the second member of (11). If both elements vanish, then (11) holds whatever C_j^1; in this case, we can consider that the second member of (11) is equal to makespan.

If several pairs of tasks satisfy these conditions, then we select the pair $\{T_i, T_j\}$ which minimizes:

$$C_j^1 - \mathbf{Min}\left(D(T_{j(1)}) + S_{j(1)}, D(T_{i+}) + S_{i+}\right) \tag{12}$$

If several pairs lead to the minimal value of (12), we select one of these pairs at random.

Finally, the algorithm used to reduce iteratively the makespan is as follows.

Algorithm CP

1. Generate a feasible solution to the problem.

   We proceed iteratively. At each iteration, we schedule at random, for each worker, the tasks which do not have unscheduled predecessors. The tasks are scheduled at the earliest.

2. Computation of the critical path.

   Obvious if we consider the definition given in 4.2.2.a. We start by identifying the task whose completion time is the makespan, then the task whose completion time is the beginning time of the previous one, and so on until we reach a task which does not have a predecessor.

3. Computation of the degree of freedom of the tasks.

   As mentioned before, this computation is conducted by applying recursively relation (10). At each iteration, we compute the degrees of freedom of $D(T_i)$ such that:

   - *either $T_{i(1)}$ does not exist or $D(T_{i+})$ has been previously computed,*

     and:

   - *either (T_{i+}) does not exist or $D(T_{i+})$ has been previously computed.*

4. Selection of the pair of tasks to be permuted.

   This is done as explained in 4.2.2.c.

   4.1. If such a pair exists, permute the tasks and return to 2.

   4.2. Otherwise, stop the algorithm.

4.3. Refinement step

This step consists of starting from the previous solution, i.e. from the solution obtained by applying H2 or CP, and of applying a classical simulated annealing approach to further reduce the makespan. Information about simulated annealing can be found in Darema et al. (1987), Johnson et al. (1989) and Kirkpatrick et al. (1983).

At each iteration of the simulated annealing process, we:

(i) choose a worker W_j at random,

(ii) choose at random two tasks T_i and T_k which are performed by W_j,

(iii) check if T_i and T_k can be permuted without violating the precedence constraints. If the answer is negative, we go back to (1), otherwise, the schedule obtained after permuting T_i and T_k is the neighbor of the previous schedule.

5. Numerical examples and evaluation of the algorithms

Two types of criteria are important to evaluate heuristic approaches, that is the computation times, and the values of the objective functions when using the heuristic approaches compared to the optimal values of the objective functions. The time criteria are easy to obtain, but the comparison of the criteria values is usually impossible since the optimal solutions of problems of reasonable size cannot be reached. In this case, one usually tries to find a lower bound to the optimal criterion. The difference between the value of the objective function obtained by using the heuristic and the lower bound is an upper bound of the difference between the value obtained from the heuristic and the optimal value.

In this paper, we propose three lower bound denoted by LB1, LB2 and LB3. LB1 and LB2 are two lower bounds to the global problem, while LB3 is a lower bound to the problem where tasks are already assigned to workers.

5.1. Lower bound LB1

This lower bound is given by relation (13).

$$LB1 = \underset{i/T_i \in E}{\mathbf{Max}} \left\{ \sum_{q=0}^{N_i} \underset{j \in \{1...,m\}}{\mathbf{Min}} p_{i(q),j} \right\} \quad (13)$$

where:

$E = \{i / i \in \{1,...,n\} \text{ and } O^{-1}(T_i) = \varnothing\}$,

$i(0) = i$, $i(q) / T_{i(q)} = O^q(T_i)$ if $q \geq 1$,

N_i is the number of successors of $T_i \in E$.

This lower bound is of good quality if a "chain" of tasks, i.e. a sequence of successors of a task $T_i \in E$, dominates all the other chains in terms of time required to perform all the tasks of the chain, assuming that their processing times are the lowest possible.

5.2. Lower bound LB2

This lower bound is given by relation (14).

$$LB2 = \left(\sum_{i=0}^{n} \underset{j \in \{1...,m\}}{\mathbf{Min}} p_{i,j} \right) \Big/ m \quad (14)$$

It is the sum of the minimal processing times distributed among the workers. It should be noticed that this lower bound is always worse than the makespan obtained by applying ASS. But LB2 is useful when the size of the problem is too important to apply ASS.

5.3. Lower bound LB3

This lower bound has been developed to evaluate the efficiency of algorithms H2 and PC. In this case, it is assumed that tasks have already been assigned to workers. LB3 is given by relation (15).

$$LB3 = \underset{j \in \{1...,m\}}{\mathbf{Max}} \sum_{i/j(i)=j} p_{i,j} \quad (15)$$

where $j(i)$ is such that T_i is assigned to $W_{j(i)}$.

5.4. Numerical examples

Several numerical examples are given in the following tables. NC is the number of precedence constraints, SA denotes the simulated annealing, M is the makespan, MW is the makespan without precedence constraints, and CT is the computation time. The other notations are those used in the previous sections. The values in bold characters are optimal makespan.

Table 1: Small-sized numerical examples

Example #		1	2	3	4	5	6	7	8	9	10	11
n		14	28	16	17	27	16	12	11	19	15	29
m		8	7	8	4	4	5	6	4	6	6	4
NC		5	8	6	7	1	7	2	3	7	7	7
H1	M	23	37	22	54	92	32	26	40	**43**	**34**	88
	CT	1"	3"	1"	1"	2"	0"	0"	0"	1"	1"	2"
ASS	MW	16	23	12	43	53	23	15	32	28	19	63
	CT	17"	1'16"	9"	30"	1'42"	16"	7"	2"	1'43"	6"	18"
H2	M	30	27	20	56	54	**25**	**19**	40	51	42	73
	CT	0"	0"	0"	0"	0"	0"	0"	0"	0"	0"	0"
SA	M	30	**23**	**17**	54	**53**	**25**	**19**	40	51	42	**63**
	CT	19"	22"	17"	21"	16"	18"	17"	15"	0"	18"	20"
CP	M	30	**23**	**17**	54	**53**	26	**19**	44	51	42	**63**
	CT	0"	0"	0"	0"	0"	0"	0"	0"	0"	0"	0"
SA	M	30	**23**	**17**	54	**53**	**25**	**19**	40	51	42	**63**
	CT	20"	0"	0"	21"	0"	1"	15"	15"	19"	18"	0"
LB1		16	17	17	40	22	25	19	32	43	34	30
LB2		9,25	20	9,25	38,25	49	19,4	10,33	26	22,67	15,5	57,75
LB3		30	19	17	43	22	25	19	40	51	42	34

Table 2: Medium-sized numerical examples

Example #		1	2	3	4	5	6	7	8	9	10	11
n		35	28	32	35	37	27	29	36	50	44	26
m		6	9	6	10	8	7	10	6	9	9	7
NC		10	6	2	8	7	11	13	14	16	15	6
H1	M	51	23	48	27	45	36	**30**	57	43	45	31
	CT	4"	4"	4"	6"	5"	2"	4"	4"	10"	7"	2"
ASS	MW	36	16	30	18	26	26	14	35	28	26	21
	CT	632"	1805"	310"	2h52'20"	3202"	1h04'22"	4754"	312"	2h25'44"	1h01'41"	208"
H2	M	46	17	**30**	20	28	34	40	40	33	32	28
	CT	0"	0"	0"	0"	0"	0"	0"	1"	0"	0"	0"
SA	M	**36**	17	**30**	**18**	**26**	34	40	**35**	**28**	27	**21**
	CT	24"	23"	30"	25"	22"	23"	26"	25"	30"	27"	22"
CP	M	38	24	**30**	25	29	34	40	36	38	30	22
	CT	0"	0"	0"	0"	0"	0"	0"	0"	0"	0"	0"
SA	M	**36**	17	**30**	**18**	**26**	34	40	**35**	**28**	28	**21**
	CT	7"	23"	0"	11"	7"	23"	26"	7"	15"	26"	6"
LB1		21	12	14	16	18	33	30	22	24	18	17
LB2		32,5	12,44	27	14,2	23,75	22,43	11	31,33	25,11	23,89	18,43
LB3		23	17	14	18	19	34	40	30	24	18	17

As we can see, heuristic H1 is very fast and sometimes provides the optimal makespan. Furthermore, H1 can be applied whatever the size of the problem. Unfortunately, in some circumstances, the solution provided by H1 is far away from the optimal one. This justifies the introduction of heuristics (ASS + H2 + SA) and (ASS + CP + SA). The problem which arises when using these heuristics is the computation time, mainly due to ASS.

6. Conclusion

We developed three heuristic algorithms which take advantage of the fact that the number of precedence constraints in a set of office tasks is low compared to the number of tasks. This fact justifies the approach which consists of solving the problem after relaxing the precedence constraints (i.e. assigning tasks to workers), and then scheduling the tasks taking into account the precedence constraints. Further researches will consist of developing new heuristics in which tasks are assigned to workers and scheduled simultaneously, taking into account the fact that the number of precedence constraints is small.

Bibliography

Cheng T.C.E. and Sin C.C.S., "A state-of-the-art review of parallel-machine scheduling research", European Journal of Operational Research, Vol. 47, pp. 271-292, 1990.

Darema F., Kirkpatrick S. and Norton V.A., "Parallel algorithms for chip placement by simulated annealing", IBM Journal of Research and Development, Vol. 31, No. 3, pp. 391-402, 1987.

Davis E., and Jaffe J.M., "Algorithm for scheduling tasks on unrelated processors", Journal of the Association for Computing Machinery", Vol. 28, No. 4, pp. 721-736, 1981.

Du J., Leung J.Y.-T. and Young G.H., "Scheduling chain-structured tasks to minimize makespan and mean flow time", Information and Computation, Vol. 92, pp. 219-236, 1991.

Graham R.L., "Bound on certain multiprocessing anomalies", Bell System Technical Journal, Vol. 45, pp. 1563-1581, 1966.

Hariri A.M.A. and Potts C.N., "Heuristics for scheduling unrelated parallel machines", Computers and Operations Research, Vol. 18, No. 3, pp. 323-331, 1991.

Hoitmont D.J., Luh P.B., Max E. and Pattipati K.R., "Scheduling jobs with simple precedence constraints on parallel machines", IEEE Control Systems Magazine, Vol. 10, No. 2, pp. 34-40, 1990.

Ibarra O.H. and Kim C.E., "Heuristic algorithms for scheduling independent tasks on nonidentical processors", Journal of the Association for Computing Machinery, Vol. 24, No. 2, pp. 280-289, 1977.

Johnson D.S., Aragon C.R., McGeoch L.A. and Schevon C., "Optimization by simulated annealing: An experimental evaluation; Part 1: Graph partitioning", Operations Research, Vol. 37, No. 6, pp. 865-892, 1989.

Kirkpatrick S., Gelatt C.D. and Vecchi M.P., "Optimization by simulated annealing", Science, Vol. 220, No. 4598, 1983.

Lenstra J.K., Shmoys D.B. and Tardos E., "Approximation algorithms for scheduling unrelated parallel machines", Mathematical Programming, Vol. 46, pp. 259-271, 1990.

Potts C.N., "Analysis of a linear programming heuristic for scheduling unrelated parallel machines", Discrete Applied Mathematics, Vol. 10, pp. 155-164, 1985.

Ullman J.D., "NP-complete scheduling problems", Journal of Computing Systems Science, Vol. 10, pp. 384-393, 1975.

Van de Velde S.L., "Duality-based algorithms for scheduling unrelated parallel machines", ORSA Journal on Computing, Vol. 5, No. 2, pp. 192-205, 1993.

Unité de recherche INRIA Lorraine
Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)

Unité de recherche INRIA Rennes - IRISA, Campus universitaire de Beaulieu 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes - 46, avenue Félix Viallet - 38031 Grenoble Cedex 1 (France)
Unité de recherche INRIA Rocquencourt - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis - 2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

ISSN 0249 - 6399